Sourcegraph

Developer onboarding: What makes it unique?

A guide to creating an effective developer onboarding program

Developer onboarding: What makes it unique?

Onboarding employees is difficult and onboarding developers is even harder.

"Only 12% of employees strongly agree that their organization does a great job onboarding new employees," according to Gallup research, meaning, in other words, that a full 88% of onboarding experiences aren't great.

As long as they eventually get onboarded, though, what's the problem? The issue is that, according to Digitate research, "employees who had a negative new hire onboard experience are twice as likely to look for new opportunities in the near future." This is particularly risky for companies trying to retain devs because, according to a CodinGame survey, the average developer considers it relatively easy—a seven on a ten-point scale—to change jobs.

Developers are both expensive and notoriously hard to recruit and hire. Imagine, then, going through all the effort of fine-tuning a developer recruiting process only to lose many developers soon after hiring them.

It's more than worthwhile, then, for organizations to take a critical look at their developer onboarding programs. That critical look starts with figuring out what makes developer onboarding different from general employee onboarding. Only with those differences in mind can companies design an effective developer onboarding program.

Table of contents

Onboarding is like a layer cake and the final layer is the most unique 1

Layer 1: Onboard developers to their corporate niches 2

Layer 2: Onboard developers to their departmental roles 3

Layer 3: Onboard developers to their functional positions 4

Example: Shopify provides a four-week getting started program 6

Example: GitLab maintains detailed engineering workflow documentation 7

Example: Sourcegraph makes its codebase accessible via shareable searches 8

Developer onboarding is essential to the developer experience 9

Onboarding is like a layer cake and the final layer is the most unique

When reassessing your developer onboarding program, it's all too easy to fall on one end of the spectrum and think that you have to either treat developers like any other employee or treat developers like a special, totally distinct type of employee. Neither is true.

Kate Gallagher, founder of Edify, a company that helps companies improve their onboarding processes, uses the guiding metaphor of a three-layer cake.

The first and top layer of onboarding is corporate; the second layer is departmental; and the third and bottom layer is functional. As you descend in the layers, onboarding becomes more nuanced and for developers, more unique.

While everyone might attend the same company culture session—a layer one activity—developers require a well-thought-out introduction to the codebase they'll be working on—a layer three activity.

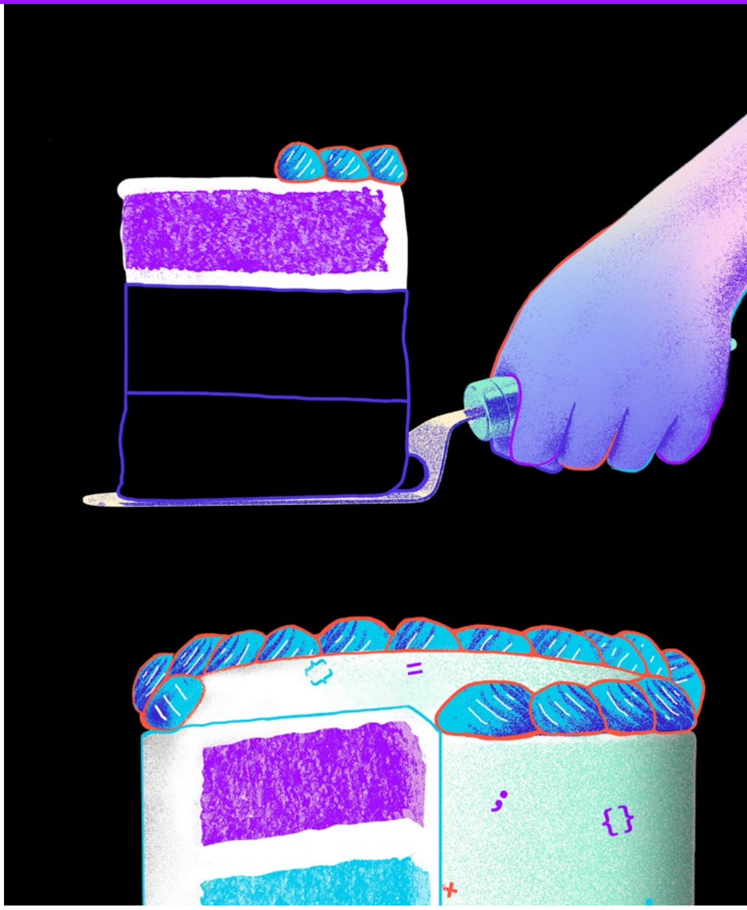
In the rest of this guide, we'll go through each layer and chart what's the same and what's different. The commonalities between general onboarding and developer onboarding will be frequent in layer one and rare by layer three. As such, layer three will include the most details. We'll also include examples of layer three-approaches from three engineering organizations.

Though we'll be going through the layers in order, the actual onboarding experience will involve "bites" from different layers throughout the onboarding process. In the same day, for example, a new developer might attend a meeting with the CEO (layer one), receive an organizational chart (layer two), and learn about their team's engineering philosophy from their engineering manager (layer three).

Layer 1:

Onboard developers to their corporate niches

In layer one, your developer onboarding program onboards developers to the organization as a whole. In this layer, onboarding developers will likely look similar to onboarding other employees.



What's similar?

The primary goal of this layer is to turn a person who was once a candidate into a fully fledged employee.

This layer, then, includes a lot of paperwork. Assuming your company is at least mid-sized, you will have an HR department or HR generalist who can handle processing these documents, some of which include:

- A form that indicates the correct amount of tax to withhold from an employee's paycheck
- A form that verifies citizenship and the legal ability to work in the relevant country
- Any insurance forms
- Any direct deposit forms
- A non-disclosure agreement

Similarly, new employees will need to get access to the company's tools and services. Access will include:

- Communication tools such as Slack or Gmail
- Project management tools such as Jira or Asana
- Expense management tools such as Expensify or Airbase
- Meeting tools such as Zoom or Google Calendar

Part of turning a candidate into an employee also involves inculcating an employee into your company culture. The processes for doing so varies between companies but might include:

- A presentation from the CEO
- A presentation from HR and/or legal
- The delivery of an employee handbook (Sourcegraph and Gitlab, for example, both provide employee handbooks that are open to the public)
- The performance of team-building activities
- The delivery of any "required reading" (Carta, for instance, requires new employees to read *The Lean Startup and How to Win Friends & Influence People*)

There's also a logistical aspect to being an employee. Before an employee's first day, you'll want them to know where to park, how they'll access the building, and where their desk is. Of course, these considerations change if your company or the position is remote, but the same principle applies.

What's different?

In this layer, only minor differences separate general employee onboarding and developer onboarding.

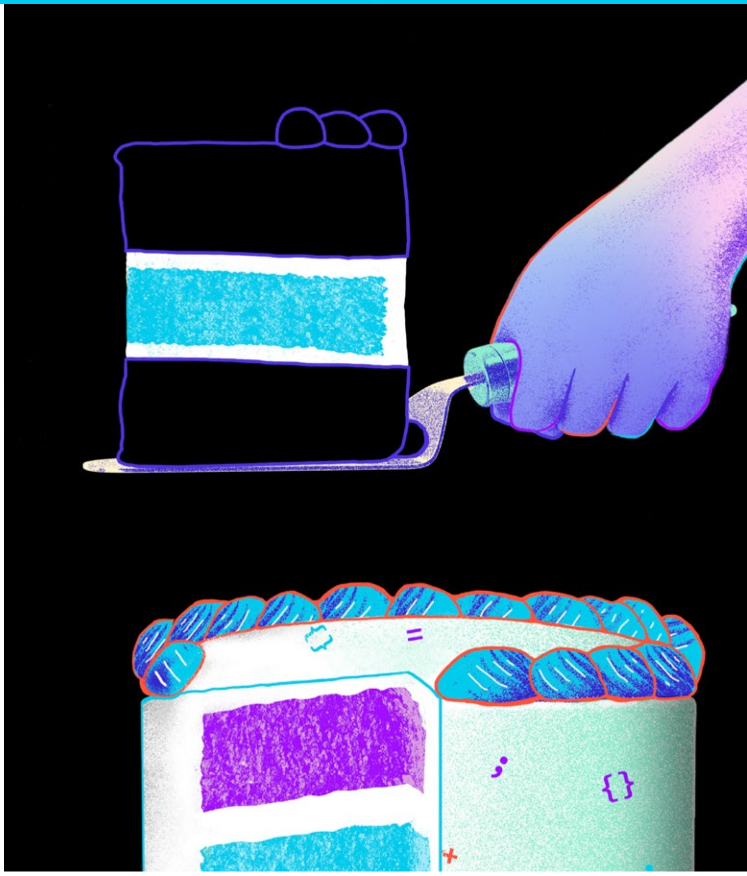
In addition to a presentation with the CEO, for instance, developers will benefit from a presentation or group discussion with the CTO.

If the company is providing an employee handbook or required reading, there may also be nuances for developers. Different sections of the handbook will be especially relevant, if not necessary (such as a section on engineering career development). Different books or articles might be provided to developers too. The Mythical Man-Month, for instance, will be relevant to developers but not to marketers.

Layer 2:

Onboard developers to their departmental roles

In layer two, your developer onboarding program onboards developers to their departments. In this layer, the developer onboarding process will include tasks and activities specific to developers but many of them will reflect things done in other departments.



What's similar?

The first layer provides new employees with a big-picture view of the organization, but employees likely won't feel that they have a place in it yet. Layer two is about situating employees in the organization via the departments they'll be working in.

One of the first things new employees will want to know is where their role sits in the department. Provide new employees with departmental and organizational charts. These charts will provide a map to employees that will help them understand their lines to other employees.

Similarly, you'll want to provide lists of other employees the new hire can contact for questions. Many of these will be role-specific, such as a software architect or project manager, but many will be role-agnostic, such as employee-facing contacts in HR, legal, and finance.

As you onboard new hires, you'll want to proactively communicate with them over and above the questions they may have for the contacts you provided.

Managers should have regular, even daily, check-ins. Similarly, it's smart for your department to encourage regular, non-work conversation across the department. New employees should meet as many people as possible in the department within their first month. Going into their first year and beyond, departments can also use programs like Donut to continue to encourage conversation.

Many organizations now also provide "onboarding buddies," teammates who are in the same department but not necessarily the same team who meet regularly with new employees and provide a space for questions and answers.

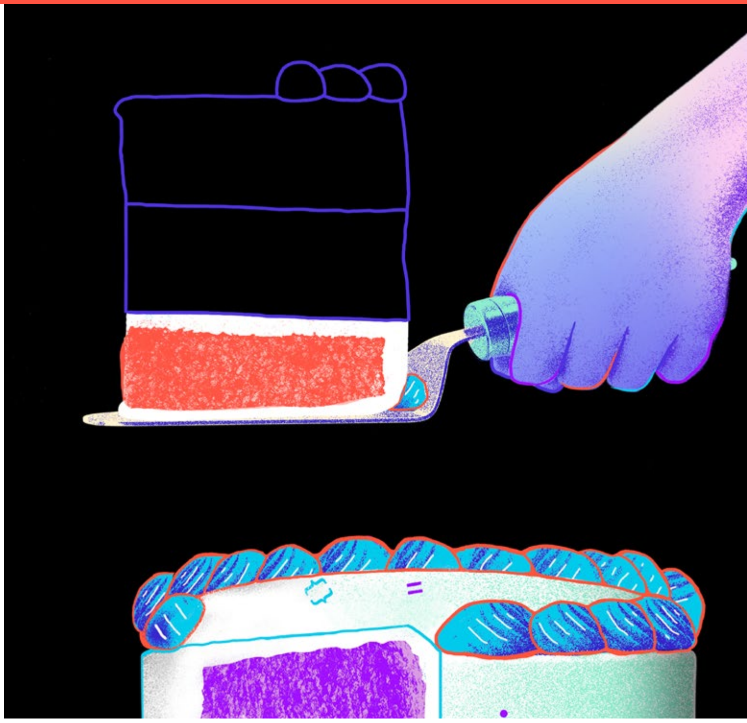
What's different?

All of the above will be similar across employees but with specific nuances for developers.

Depending on the size of your organization and the complexity of the engineering department, the organizational charts you provide developers may also be different. While other employees may report primarily, if not exclusively, to their immediate supervisor, developers may also report to project and product managers.

Onboarding buddies, too, are a little different for developers. Ideally, an onboarding buddy for a new developer is technical and more experienced. Many of the questions an onboarding developer will have will be technical in nature and the most useful, effective onboarding buddy will have a skillset they can use to provide the new developer with answers.

Layer 3:
Onboard developers to their functional positions



In layer three, your developer onboarding program onboards developers to their functional positions and teams. Gallagher refers to this layer of the cake as “your most specific and most important and hopefully tastiest layer.”

You can muddle through by copy-and-pasting much of the onboarding processes from employee to employee in layers one and two, but if you do the same in layer three, you will delay developer productivity, slow team velocity, and encourage attrition.

Note that, depending on the organization, layers two and three may blur. In some organizations, departments are organized with different developers having different functional roles, but in other organizations, all the developers in a given department have the same functional role. That said, the framework and its principles still apply.

What’s similar?

The aspects of onboarding in this layer are only nominally similar between employees.

For everyone, you’ll want to provide basic software tutorials. Developers, however, may need more tutorials, because they use more software, and more robust tutorials, because that software can be more complex. These tutorials, however, may only need to include advanced tips or team-specific workflow guides because you may choose to only hire developers who are already familiar with, say, GitHub and Jira.

You’ll also want to provide all employees with a workstation, whether that be a desktop and a cubicle or table, if in person, or a laptop, if remote. Developers, however, will likely require machines with specific specs and multiple monitors.

What’s different?

One way to think through this layer is to think through a set of core questions. Your goal here is to identify tacit knowledge and make it explicit and learnable.

Gallagher, for instance, recommends asking:

- What’s your engineering philosophy?
- What’s your software development lifecycle like?
- What are your tool systems and toolsets?
- How do you actually do your work?

The answers to these questions will be the foundation on which to build your developer onboarding program.

Many organizations are already building on this foundation via developer experience or platforms teams. These teams have the explicit goal of making the developer experience as smooth and productive as possible. Their work extends from ensuring a developer’s local development environment is properly set up to troubleshooting build errors.

At Sourcegraph, we have a developer experience team and a #dev-experience Slack channel where developers can ask questions and the team can announce quality-of-life improvements.

It’s in this layer, and in dealing with these teams, that terms like “developer velocity” start to emerge. Developers are expensive and “time-to-productivity” is an important metric to track.

Time-to-productivity, however, is a notoriously difficult metric to measure. The last thing you want to do is count how many lines of code a developer merged or how long a developer worked on a given PR. These measures are not only inaccurate but misleading. Instead, you'll want to ask questions like:

- How long does it take a developer to start working on their own?
- How long does it take for committed code to be mostly or completely bug-free?
- How long until the volume of PRs committed by the new developer is relatively similar to the volume of PRs committed by a teammate at the same level?
- How long does it take for other developers to review the new developer's PRs?
- How many tickets can a new developer close, on average, in a given sprint?

This list is non-exhaustive. Time-to-producitivy is made even thornier by a seemingly industry-wide optimism.

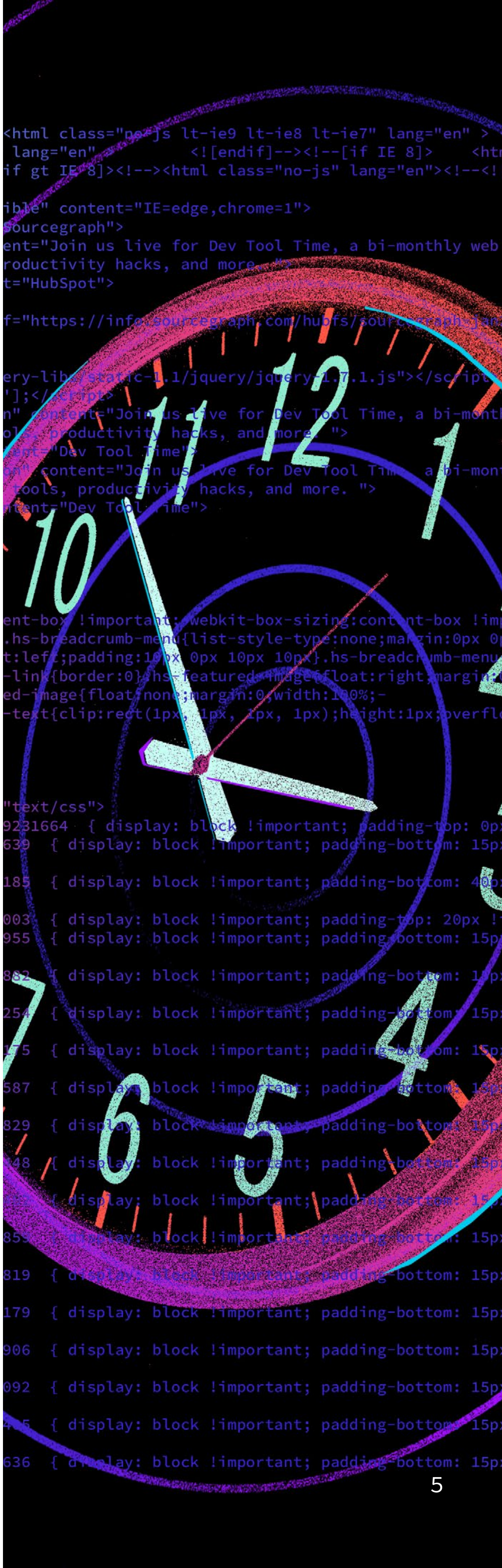
Gallagher notes that, in her experience, developers usually need between nine and twelve months to be productive but most companies predict it will take between three and six months. A good developer onboarding program, in her eyes, can bring time-to-productivity down to three months. But this speed is only possible if your company acknowledges and works from the gap that likely exists between how fast you want to be and how fast you actually are.

You'll want to make sure the team your new hire is joining can provide:

- Documentation for the project they'll be working on
- Documentation for the toolset they're going to use
- A list of common bugs in their project or workflow and how to fix them
- A standardized procedure for setting up a local development environment
- Documentation that explains how to use your codebase, as well as a design doc that explains how the codebase has been designed and the core principles behind its design

The nuances in layer three make all the difference between a great developer onboarding program and a middling one. It's worthwhile to invest in a developer experience team who can consider it a core priority to fine-tune the technical aspects of developer onboarding, ensuring developer productivity is within reach on day one.

Layer three also provides the most opportunity for experimentation and innovation. What we've covered so far is only the basics. To have a truly great developer onboarding program, one that leverages the uniqueness developers, and your company, benefits from, you'll have to reach further. To illustrate, we've added three examples of companies with developer onboarding processes that go beyond the basics.



Example:
Shopify

Shopify provides a four-week getting started program

Shopify anchors its developer onboarding experience around a four-week program, according to Jerie Shaw, Senior R&D Education Program developer at Shopify.

In the first week, developers learn about Shopify's history and culture—a layer one activity—and do basic laptop setup—a layer two activity. Shopify, as well as other modern technology companies, prefers developers to ship within their first week—a layer three activity.

In the second week, new developers work with the support team. Shopify requires this so that developers get a hands-on perspective with the customer experience. According to Shaw, they'll build a test store, configure shipping settings, customize theme code, and do anything else that will give them an intimate understanding of the Shopify product. Meanwhile, they'll answer actual emails from merchants and help those merchants figure out problems with their stores.

In the third and fourth weeks, new developers will work through a custom onboarding plan built by their managers. Shopify's onboarding experience is advanced enough that the company has numerous learning experiences to offer, such as bootcamps, workshops, and self-directed activities. Managers can select from these experiences and construct a plan that the new developer works through in the coming weeks. Weeks three and four, as well as the following weeks, then look very different depending on the manager and team.

Shopify, even though it's fully remote, believes in synchronous work. In weeks one and two, new developers are working in a deliberate structure and largely working within a group and at the same time. In weeks three and four and beyond, according to Shaw, they "start a gradual process of weaning off of structured learning and workshops, towards more problem-based learning, more team-based learning, more project-based learning."

Example:
GitLab

GitLab maintains detailed engineering workflow documentation

GitLab values asynchronous work As such, GitLab provides copious amounts of documentation, all of which enables new developers to learn much, if not all, of what they need to know. And they can learn all this without ever having to have a synchronous meeting.

GitLab's developer onboarding page is a hub of links leading to pages on environments, infrastructure, development, and more. Of particular interest is this lengthy page on their engineering workflow, which describes a layer three activity.

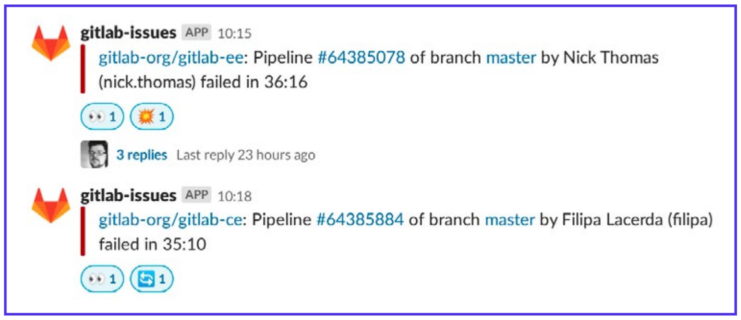
In this document, GitLab lays out the workflow that anyone working with issues at GitLab will use, including:

- GitLab Flow, which describes a set of best practices that "combines feature-driven development and feature branches with issue tracking."
- Code Review Guidelines, which describes the values behind the code review process and explains the reviewer and maintainer roles.
- Security Issues, which describes the basics of handling a security issue as well as how to work with the GitLab security team.

Let's say a new developer breaks [master]. That new developer, without asking anyone for help, can go to this section in the engineering workflow documentation to figure out:

- What is a broken master?
- Broken master service level objectives
- Broken master escalation
- Triage broken master
- Resolution of broken master
- Merging during broken master
- Broken master mirrors

GitLab goes so far as to provide an example of what the reactions to a failure look like in Slack.


gitlab-issues APP 10:15
gitlab-org/gitlab-ee: Pipeline #64385078 of branch master by Nick Thomas (nick.thomas) failed in 36:16

3 replies Last reply 23 hours ago

gitlab-issues APP 10:18
gitlab-org/gitlab-ce: Pipeline #64385884 of branch master by Filipa Lacerda (filipa) failed in 35:10

With documentation in hand, new developers can onboard themselves and GitLab can minimize the amount of synchronous onboarding senior developers need to do.

Example:
Sourcegraph

Sourcegraph makes its codebase accessible via shareable searches

It's no surprise that we use Sourcegraph to onboard new developers at Sourcegraph. We do so primarily in the third layer, where the most help is needed. One of the five primary use cases of the Sourcegraph code intelligence platform, after all, is developer onboarding.

A few of the ways developers use Sourcegraph to accelerates and improves our own developer onboarding include:

- New developers at Sourcegraph can use code search to find answers to their questions in our codebase, rather than monopolizing the time of senior developers with repetitive questions.
- Engineering managers can make our codebase nearly self-serve, enabling all of our developers to navigate with definitions and references.
- Experienced developers can use Notebooks to document actionable, shareable codebase searches.

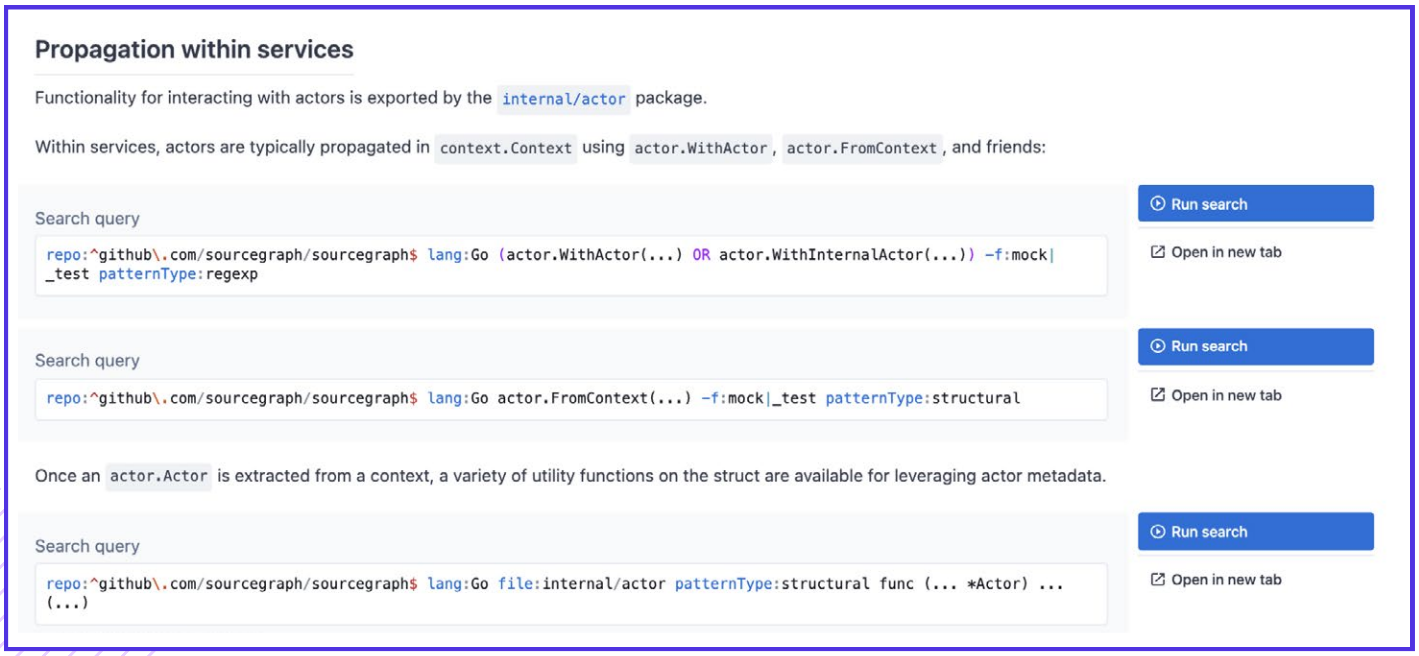
The last item is rapidly becoming a core part of Sourcegraph's developer onboarding program.

A new developer at Sourcegraph might, for example, not know what actors represent. Rather than asking another developer to explain, new developers can view this Notebook. This Notebook describes actors as representing agents that access resources and explains that these actors could be anonymous users, authenticated users, or internal Sourcegraph services.

Past the text, the Notebook provides pre-filled searches that new developers can run, from within the Notebook, to understand propagation within services and propagation between services.

At the bottom, developers will also find links to the sourcegraph#27918 issue as well as the sourcegraph#27916 issue where that work originated. New developers can recover context and perform codebase searches from one centralized place, ensuring they can understand this concept, among many others, first-hand.

Developers at Sourcegraph are building similar Notebooks every day and have so far discovered compounding benefits to making our codebase more accessible.



Propagation within services

Functionality for interacting with actors is exported by the `internal/actor` package.

Within services, actors are typically propagated in `context.Context` using `actor.WithActor`, `actor.FromContext`, and friends:

Search query

```
repo:^github\.com/sourcegraph/sourcegraph$ lang:Go (actor.WithActor(...) OR actor.WithInternalActor(...)) -f:mock|
_test patternType:regexp
```

Search query

```
repo:^github\.com/sourcegraph/sourcegraph$ lang:Go actor.FromContext(...) -f:mock|_test patternType:structural
```

Once an `actor.Actor` is extracted from a context, a variety of utility functions on the struct are available for leveraging actor metadata.

Search query

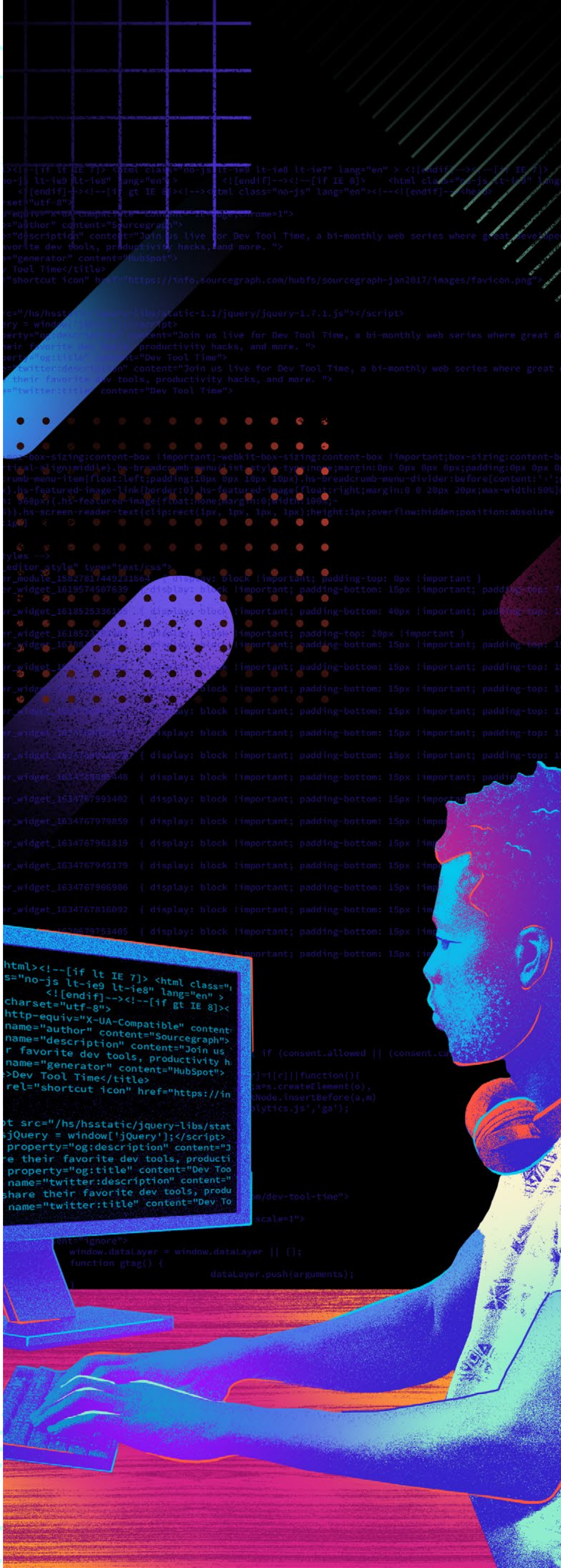
```
repo:^github\.com/sourcegraph/sourcegraph$ lang:Go file:internal/actor patternType:structural func (... *Actor) ...
(...)
```

Developer onboarding is essential to the developer experience

You are always already onboarding. The tiniest startup and the biggest enterprise and all the companies in between onboard—the difference is whether onboarding is done well and whether it's done intentionally.

Your developer onboarding program might be the most important onboarding program in your company. If you don't onboard a developer well, it's relatively easy for them to get a similar, high-paying job elsewhere—leaving you to replace them.

That said, don't build your developer onboarding program from a place of fear. Hiring and onboarding a new developer is an exciting opportunity to empower a new employee and provide support for an entire team or department. You can best serve the opportunity each new developer provides with a great developer onboarding program.



Sourcegraph

about.sourcegraph.com

Sourcegraph is a code intelligence platform that unlocks developer velocity by helping engineering teams understand, fix, and automate across their entire codebase.